Altix Tips

Samson Cheung
Cheung@nas.nasa.gov

Topics

- The Columbia
- Memory issues
- Unsupported compiler flags

The Columbia

- Intel Itanium 2 Rev 5 processor, 1.5GHz
- Front-end (columbia): 64 CPUs
- 20 boxes, each has ~1000 GB:
  - Columbia1, …, Columbia20: 512 CPUs
  - Columbia20 has 32 CPUs with 1.6GHz
- Columbia1-12
  - 4 IX-Bricks, 128 C-Bricks, 112 R-Bricks
- Columbia13-20
  - 4 IX-Bricks, 64 C-Bricks, 48 R-Bricks

Memory Issues

First Touch Policy

- By default, all pages are allocated with a “first touch” policy
- Always initialize data with the “first-touch” policy with multiple processors in a parallel loop
  - Data is distributed naturally
  - Each processor has local data
  - Minimal data exchange between nodes
  - Page edge effects

dplace

- dplace binds processes to specified CPUs in round-robin fashion; once pinned, they do not migrate (a la IRIX _DSM_MUSTRUN).
  - `c <cpulist>` CPU numbers are logical numbers relative to current cpumemset.
  - `x <mask>` A bitmask for specifying threads to skip placing. [See following examples.]
  - `s <count>` Skip placement of the first <count> threads**.** Use –`s1` to skip placing the shepherd thread in MPI programs.
  - `q` Displays static load information. `dplace` without arguments will avoid loaded cpus.
  - `E` Exact placement

I only know …

- ProPack 2.4 OpenMP: dplace –x6 …
- ProPack 3 OpenMP: dplace –x2 … (?)
- `'setenv LD_ASSUME_KERNEL 2.4.19'` in ProPack 3 to revert to old Linuxthreads behavior

Memory Usage

- dlook
  - identifies distribution of application memory pages across nodes (dlook my.exe)
- /proc/discontig file
  - be aware of per-node memory availability
  - cat /proc/discontig
  - In I/O intensive environment, Linux gladly eats up available memory for I/O buffer cache

Fortran Memory Management

- Intel® 7.1 Fortran runtime libraries have their own memory management routines for handling automatic arrays, allocatable arrays, etc.
- At least some of these rely on mmap()/ munmap()
- Can lead to poor scaling of parallel codes
- Workarounds:
  - Compile with `-stack_temps` flag
  - Use Cray pointers, which are malloc-based
- These issues are resolved in 8.0 compilers

Bcopy/Memcpy on Altix™

- Standard glibc bcopy/memcpy routines are slow
- MPT has optimized bcopy routine, fastbcopy
  - For best performance, source and destination addresses should be word-aligned and transfer length should be a multiple of 8 bytes
- For non-MPI codes can access fast bcopy routine via SCSL's __scsl_bcopy (not tried)

Unsupported Flags

Unsupported Compiler Flags

- Categories
  - mP2OPT_*: HLO (high-level optimizer), loop nest optimization, prefetching, profile-guided optimization, etc.
  - -mP3OPT_*: Code generation, pipelining, load latencies, etc.
  - -mPAROPT_*: OpenMP and automatic parallelization controls
  - -mIPOPT_*: Interprocedural optimization knobs
- floating point / Integer loads latency (6-11 cycles)
  - For increasing latency, use `-mP3OPT_ecg_mm_fp_ld_latency=##` (`-mP3OPT_ecg_mm_int_ld_latency=##` for integer loads)
  - MIPSpro analogue is `-CG:ld_latency=##`

Prefetching

- `-mP2OPT_hlo_prefetch=F`
  - Compiler sometimes prefetches too aggressively
  - Cache-contained data doesn't require prefetching; lfetch operations consume instruction issue slots
  - If `–O2` gives better performance than `–O3`, it may be due to this (prefetching is enabled only at `–O3`)
- MIPSpro equivalent: `-LNO:prefetch=OFF`

Prefetching (Conts.)

- `-mP2OPT_hlo_pref_hint=#`
  - 0: no prefetch hint (temporal locality at all cache levels, very bad for floating point data)
  - 1: nt1 hint (no temporal locality at cache level 1)
  - 2: nt2 hint (no temporal locality at level 2)
  - 3: nta hint (no temporal locality at any level)
- Hint applies to all prefetch instructions throughout file -- if source code can be modified, may be better to insert mm_prefetch() calls (never tried)
- MIPSpro analogue: -LNO:pf1=[ON|OFF]:pf2=[ON:OFF]

Prefetching (Conts.)

- `-mP2OPT_hlo_level=##`
  - -1: HLO code generation without optimization
  - 0: disable HLO and SWP
  - 1: perform all HLO optimizations (default)
  - 2: prefetch only
- using `-mP2OPT_hlo_level=2`
- No single equivalent MIPSpro flag (-LNO:fission=OFF:fusion=OFF:blocking=OFF:…)

Loop interchange and jamming

- Check `-opt_report` output for information on loops that the compiler interchanges
- Disable loop interchange using
  - `-mP2OPT_hlo_linear_trans=F`
- MIPSpro equivalent: `-LNO:interchange=OFF`
- 2D loop unroll and jam is disabled by default (in version 7, don't know in current 8.0 releases)
- Enable 2D loop unroll and jam with
  - `-mP2OPT_hlo_loop_unroll_jam=T`

Linpack Performance Report

- Flags used in Linpack 100 score (1659 Mflops):
- Flags used: `-O3 -ipo -fno-alias -mP2OPT_hlo_loadpair=F -mP2OPT_hlo_prefetch=F -mP2OPT_hlo_loop_unroll_factor=2 -mP3OPT_ecg_mm_fp_ld_latency=8`
  - `-mP2OPT_hlo_loadpair=F`: disable generation of floating-point load pair instructions
  - `-mP2OPT_hlo_loop_unroll_factor=2`: unroll all loops by 2
- Without hidden flags performance drops by 33%

Profiling Tools

Some of Them

- pfmon
- profile.pl
- histx
- lipfpm

lipfpm

- For help: lipfpm –h
  - Can specify up to 4 events at a time
  - -f is required for MPI codes
- Collective events (-c bw)
  - counters associated with (read) bandwidth
- lipfpm –c bw my.exe

profile.pl

- For help: man profile.pl
- A perl script uses pfmon
- mpirun –np 8 profile.pl –QS -s1 -c0-7 ./my.exe
- Flags -c, -s, -n, -p are same as dplace